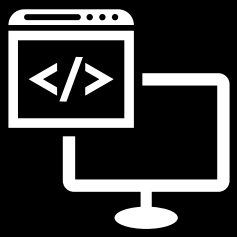
PROJET ANALYSE DE DONNÉES PYTHON

PERROT Loick



Ce projet nécessite de mettre en avant nos compétences en affichage visuel à travers plusieurs graphiques sur le logiciel Thonny à partir de la base de données qui nous a été fournis sous formats json. En effet, le département d'agriculture des États-Unis nous a partagé un fichier nommé "usda-food-database.json" qui nous donne des informations sur la composition nutritionnelle détaillée de divers aliments. Ce jeu de données indique donc le nom de l'aliment, sa portion, ses nutriments, micro-nutriments ainsi que des détails biochimiques. Pour mener à bien ce projet, nous avons décidé de faire recours à certains modules et librairies tels que :

- Pandas : Librairie très utile pour l'analyse de données. Utilisée ici pour lire les fichiers txt, regrouper toutes les données dans un seul data frame, mais aussi pour filtrer et regrouper les données, ainsi que pour réaliser des calculs (somme, moyenne...).

- Os : Module utilisé ici uniquement pour la préparation des données. On s'en sert pour vérifier l'existence du dossier "names", mais aussi pour lister son contenu et pour construire les chemins entiers des fichiers (chemin jusqu'à names + nom du fichier.txt).

- Tkinter : Librairie servant à créer l'interface constituée d'une fenêtre principale contenant le premier graphique souhaité ainsi que tous les éléments de bases.

- Tkinter.ttk : Sous-module de Tkinter permettant d'introduire des cadres/panneaux ainsi que des listes déroulantes pour changer de graphique et/ou pour interagir avec l'utilisateur (par exemple : choix de l'année).

- Messagebox : Sous-module de Tkinter également qui sert simplement à renvoyer des messages d'erreurs en cas de problèmes d'exécution du code.

- Json : Module que l'on utilisera dans notre fonction charger_donnees() afin de transformer le contenu textuel du fichier .json en une liste d'objets Python que le programme pourra ensuite parcourir pour extraire les calories, les protéines, etc.

Nous avons créé un code permettant d'avoir une interface graphique dans laquelle nous retrouvons les analyses demandées telles que :

- Les 20 nourritures les plus et les moins caloriques.

• Les 20 meilleurs et pires selon plusieurs informations de nutrition (lipides (lipid), glucides (carbohydrate), protéines (protein), etc.).

• Les 20 meilleurs et pires aliments sur les différentes vitamines.

• Le classement des groupes de nourritures selon les données de nutritions.

• Une analyse par groupe et par aliment d'un nutriment sélectionné.

À présent, nous allons passer à la préparation des données qui s'est dans un premier temps fait avec le dictionnaire NUTRIENTS_TO_EXTRACT puis avec la fonction charger_donnees. Voici comment nous avons procédé :

Le dictionnaire NUTRIENTS_TO_EXTRACT sert à dire sous quelle forme on souhaite présenter nos données dans le tableau final. Par exemple, pour 'Énergie (Kcal)': ('Energy', 'kcal') on a à gauche la clé = nom propre choisi et à droite l'instruction qui va dire à Thonny d'aller chercher la donnée qui s'appelle "Energy" et dont l'unité vaut "Kcal".

Analyse de la fonction charger_donnees :

D'abord on récupère notre variable DF_NUTRI défini avant de créer la fonction (DF_NUTRI = None) qui sera notre future data.frame regroupant toutes les données prêtes à être exploitées.

Puis nous ouvrons le fichier usda-food-database.json en vérifiant au préalable que le chemin qui nous y mène existe bien. On charge ensuite les données en utilisant json.load afin de transformer le texte brut en une liste d'objets Python.

À travers une boucle for, nous récupérons toutes les informations de base pour chaque aliment c'est-à-dire son nom (description) et sa catégorie (group) que nous mettons dans une variable food_info. Puis nous mettons dans une variable nutrients_map la valeur (value) de chaque nutriment suivant leur nom (description) et leur unité (units). Et pour chaque donnée de nutrients_map on récupère sa valeur (value) que l'on va convertir en nombre décimal (avec float) pour rajouter cette valeur à la variable food_info qui est maintenant composé du nom et de la catégorie de l'aliment ainsi que de la valeur de ses nutriments (avec les nutriments présentés comme dans NUTRIENTS_TO_EXTRACT).

Toutes les lignes de food_info sont ajoutées à une liste appelée all_foods et quand on fait pd.DataFrame(all_foods), Pandas prend cette liste de dictionnaires et les aligne pour créer le tableau final (DF_NUTRI) qu'on utilisera pour nos classements.

On créé une variable groupes_dispo dans laquelle on a trié par ordre alphabétique les noms de la colonne groupe du data.frame DF_NUTRI en supprimant les doublons. On injecte cette liste triée dans le menu déroulant de l'interface graphique et par défaut on sélectionne le premier

groupe de la liste. var_analyse.set('Énergie (Kcal)') sert à donner un point de départ automatique à notre application avant d'appeler notre fonction mettre_a_jour_tout qui, comme on le verra après, sert à mettre à jour tous les tableaux en fonction du nutriment sélectionné.

Et enfin on nomme la fenêtre "Analyseur USDA Complet - {len(DF_NUTRI)} aliments - {len(groupes_dispo)} groupes" avec {len(DF_NUTRI)} et {len(groupes_dispo)} le nombre d'aliments et de groupes analysés.

S'il y a un quelconque problème (appelé e) lors de l'exécution de la fonction, alors on renvoie le message d'erreur suivant : "Erreur critique", f"Impossible de lire le fichier : {e}".

Les autres fonctions que nous allons voir à présent (avant de passer à l'interface graphique) servent à préparer nos tableaux :

Fonctions pour gérer les tableaux

Au lieu de copier-coller le code de création/gestion de tableau pour chaque onglet (Top global, Pires globaux, Classement groupes, Détail groupe), nous avons centralisé ça dans des fonctions. Cela rend le code plus propre et plus facile à modifier. **A . La création de la structure : creer_tableau**

Cette fonction sert à créer des tableaux sur commande. Au lieu de copier le code de configuration pour chaque onglet (Top, Flop, Groupes, Détails), nous appelons cette fonction qui retourne un objet prêt à l'emploi.

Voici comment elle procède :

- **Initialisation du widget :** Nous créons un objet ttk.Treeview. Par défaut, ce widget est conçu pour présenter des structures arborescentes (type explorateur de fichiers) et réserve systématiquement une première colonne (identifiée comme #0) pour afficher l'icône de déploiement des nœuds hiérarchiques. Étant donné que nos données sont ne sont pas structurées sous forme de listes hiérarchiques, cette colonne par défaut était inutile. L'option show="headings" nous permet de la désactiver, pour forcer le widget à se comporter comme un tableau, n'affichant que les colonnes de données définies ("Nom" et "Valeur") sans la marge réservée à l'arborescence.

- **Configuration des colonnes :** Nous définissons les entêtes ("Nom" et "Valeur") et nous ajustons la largeur des colonnes et on centre les données dans la colonne « valeur » pour un rendu plus esthétique.

- **Système de défilement :** Étant donné que nos listes peuvent contenir des milliers d'aliments, une barre de défilement (Scrollbar) est indispensable. La barre contrôle la

vue verticale du tableau (command=tree.yview) et le tableau met à jour la position du curseur de la barre (yscroll=scrollbar.set).

• **Affichage :** Enfin, la fonction retourne l'objet tree créé afin qu'il puisse être manipulé par la suite.

B . Le remplissage du tableau : remplir_tableau

Une fois la structure vide créée, cette fonction se charge d'injecter les données issues de nos analyses Pandas à l'intérieur du tableau. Elle est appelée à chaque fois que l'utilisateur change de nutriment ou de groupe. Son fonctionnement se décompose en trois étapes :

• **Nettoyage :** Avant d'ajouter de nouvelles données, la boucle for item in tree.get_children(): tree.delete(item) vide intégralement le tableau. Cela évite d'empiler les résultats les uns sur les autres lors des mises à jour successives.

• **Vérification de sécurité :** Nous vérifions si le *DataFrame* fourni est valide ou vide pour éviter de faire planter l'application inutilement.

• **Itération et Formatage :** Nous parcourons le *DataFrame* ligne par ligne grâce à .iterrows(). Avant l'insertion, nous appliquons un formatage aux valeurs numériques (f" {val:.2f}") pour limiter l'affichage à deux décimales, rendant la lecture des nutriments plus claire (ex: afficher "12.50" au lieu de "12.50342"). Enfin, la méthode tree.insert ajoute la ligne nettoyée dans le tableau.

La fonction mettre_a_jour_tout :

Fonction qui sert simplement à actualiser les fonctions suivantes.

La première ligne récupère le nutriment sélectionné qu'on met dans la variable nutriment.

La seconde ligne sert à mettre à jour l'onglet général à partir de ce nutriment grâce à la fonction analyser_global.

La troisième met à jour le classement des groupes avec la fonction analyser_classement_groupes qui récupère également le nutriment choisi.

Et la quatrième ligne met à jour l'analyse spécifique par groupe (si un groupe est sélectionné) grâce à la fonction analyser_groupe_specifique. Nous allons donc regarder comment marchent ces fonctions.

La fonction analyser_global :

Fonction grâce à laquelle nous pouvons réaliser nos différents top 20 (calorie, informations de nutritions, vitamines…). Elle prend comme paramètre le nutriment sélectionné, puis elle met en place le top 20 des aliments dans lesquels on retrouve le plus de ce nutriment. Pour cela elle tri la colonne du nutriment dans l'ordre décroissant avant de prélever les 20 premières lignes que l'on met dans le data.frame df_top et on peut alors remplir le tableau de gauche (grâce à tree_top_global que l'on verra plus tard) avec notre fonction remplir_tableau.

On fait la même chose pour les 20 aliments qui contiennent le moins ce nutriment en rangeant cette fois-ci par ordre croissant.

Et on met à jour le titre de l'encadré des tableaux (lbl_frame_top défini plus tard).

La fonction analyser_classement_groupes :

Cette fonction servira à remplir le tableau de gauche dans l'onglet "Analyse par Groupes" qui représente le classement des groupes de nourritures selon les données de nutritions. Dans cette fonction, on fait d'abord la moyenne du nutriment choisi pour chaque groupe que l'on va mettre dans le data.frame df_groupes et dans ce nouveau data.frame on range par ordre décroissant nos valeurs que l'on insère dans df_groupes_ties.

Il ne nous reste plus qu'à appeler la fonction remplir_tableau avec notre dernier data.frame pour avoir le tableau qui renvoie le classement de la quantité moyenne de nutriment par groupe. Et on finit par nommer l'encadré du tableau qui se mettra automatiquement à jour.

La fonction analyser_groupe_specifique :

Fonction qui permet de faire l'analyse par groupe et par aliment d'un nutriment sélectionné. Elle travaille sur la partie "Analyse par Groupes" de l'interface pour laquelle on récupère dans un premier temps le nutriment et le groupe sélectionné. Puis on filtre DF_NUTRI en ne gardant que les données pour lesquelles le groupe correspond à celui choisi (que l'on met dans df_filtre) avant de trier par ordre décroissant pour récupérer les 50 plus grandes valeurs soit les 50 aliments qui ont la plus grande quantité du nutriment choisi. Ces 50 lignes sont stockées dans df_tri qui sera utilisé pour créer le tableau de droite.

Création de l'interface graphique avec tkinter :

L'interface a été conçue pour être simple à utiliser : on sépare les commandes (en haut) des résultats (les tableaux). Voici comment nous l'avons construite étape par étape :

1. Création de la Fenêtre Principale

Tout commence par la création de la fenêtre de l'application (l'objet root).

- Nous lui donnons un titre : "Analyseur de Nutrition".
- Nous fixons sa taille à 1200x700 pixels pour avoir assez de place pour afficher les grands tableaux.

2. La Barre de Contrôle (Le bandeau du haut)

tk.Frame(...) : On crée une "boîte" rectangulaire vide. C'est un conteneur qui sert juste à regrouper le texte et la liste déroulante ensemble.

.pack(fill=tk.X) : On colle cette boîte tout en haut de la fenêtre et on lui dit de s'étirer sur toute la largeur (fill=tk.X) pour faire une barre complète.

tk.StringVar() : C'est une "mémoire" spéciale pour l'interface. Elle sert à stocker le texte choisi par l'utilisateur.

ttk.Combobox(...) : C'est la liste déroulante. On la connecte à la mémoire var_analyse (pour savoir ce qui est écrit dedans) et on lui donne la liste des nutriments (values=...).

.bind(...) : sert à déclencher la fonction mettre_a_jour_tout quand l'utilisateur sélectionne un objet de la liste déroulante.

3. Le Système d'Onglets

Pour ne pas tout mélanger sur le même écran, nous utilisons un système de pages, comme sur un navigateur web (appelé Notebook). Cela nous permet de créer deux espaces distincts.

ttk.Notebook(root) : On crée un système d'onglets, comme un classeur physique.

notebook.add(tab_general, text="Top...") : On ajoute des pages (des Frame) à ce classeur et on écrit le titre sur l'étiquette de l'onglet.

4. L'Onglet 1 : Les Tops et les Flops

Cette page sert à voir les extrêmes. Nous divisons l'écran en deux parties :

• À gauche (Cadre vert) : Pour afficher les 20 meilleurs aliments.

• À droite (Cadre rouge) : Pour afficher les 20 aliments les plus faibles. Pour créer ces tableaux rapidement sans réécrire de code, nous utilisons notre fonction creer_tableau que nous avons fabriqué plus tôt.

tk.LabelFrame(...) : C'est un cadre avec une bordure et un titre intégré. « fg=... » sert à donner une couleur au texte. creer_tableau(...) : Ici, on gagne du temps. Au lieu d'écrire 10 lignes de code pour créer le tableau, on appelle notre fonction creer_tableau que nous avons créé plus tôt qui nous fabrique un tableau prêt à l'emploi.

On répète ce processus 2 fois pour créer les tableaux du top 20 et des 20 pires aliments

5. L'Onglet 2 : L'Analyse par Familles

Cette page est un peu plus complexe et se divise aussi en deux :

• Partie Gauche : Nous créons un tableau manuel pour afficher la moyenne de chaque groupe. Ici, nous ne pouvions pas utiliser notre raccourci habituel car les noms des colonnes sont différents ("Groupe" au lieu de "Nom"). Le système pour créer le tableau reste le même que pour la fonction creer_tableau faite plus haut et on utilise les données extraites des fonctions d'analyse créées avant.

• Partie Droite : Nous ajoutons une deuxième liste déroulante pour choisir une famille précise (ex: "Produits laitiers"). Juste en dessous, un tableau affiche uniquement les aliments de cette famille.

6. Le Lancement de l'Application

Pour finir, le code contient deux ordres importants :

1. Le chargement intelligent : On demande à l'application d'attendre une fraction de seconde (root.after) avant de lire le gros fichier de données. Cela permet à la fenêtre de s'afficher immédiatement, sans donner l'impression de bloquer au démarrage.

La boucle infinie : La commande root.mainloop() permet de garder la fenêtre ouverte et d'attendre que l'utilisateur clique quelque part. Sans ça, le programme s'arrêterait tout de suite.